(19) Europäisch s Patentamt
European Patent Offic
Office européen d s brevets

(11) EP 0 747 816 A2

(12) # EUROPEAN PATENT APPLICATION

(43) Date of publication:
11.12.1996 Bulletin 1996/50

(51) Int Cl.[6]: **G06F 9/46**

(21) Application number: **96480079.1**

(22) Date of filing: **31.05.1996**

(84) Designated Contracting States:
DE FR GB

(30) Priority; **07.06.1995 US 473692**

(71) Applicant: **INTERNATIONAL BUSINESS MACHINES CORPORATION**
Armonk, NY 10504 (US)

(72) Inventors:
- **Eickemeyer, Richard James**
  Rochester, MN 55901 (US)
- **Johnson, Ross Evan**
  Rochester, MN 55906 (US)
- **Kossman, Harold F.**
  Rochester, MN 55906 (US)
- **Kunkel, Steven Raymond**
  Rochester, MN 55901 (US)
- **Mullins, Timothy John**
  Rochester, MN 55901 (US)
- **Rose, James Allen**
  Rochester, MN 55901 (US)

(74) Representative: **Lattard, Nicole**
Compagnie IBM France
Département de Propriété Intellectuelle
06610 La Gaude (FR)

(54) **Method and system for high performance multithread operation in a data processing system**

(57) A method and system for enhanced performance multithread operation in a data processing system which includes a processor, a main memory store and at least two levels of cache memory. At least one instruction within an initial thread is executed. Thereafter, the state of the processor at a selected point within the first thread is stored, execution of the first thread is terminated and a second thread is selected for execution only in response to a level two or higher cache miss, thereby minimizing processor delays due to memory latency. The validity state of each thread is preferably maintained in order to minimize the likelihood of returning to a prior thread for execution before the cache miss has been corrected. A least recently executed thread is preferably selected for execution in the event of a nonvalidity indication in association with all remaining threads, in anticipation of a change to the valid status of that thread prior to all other threads. A thread switch bit may also be utilized to selectively inhibit thread switching where execution of a particular thread is deemed necessary.

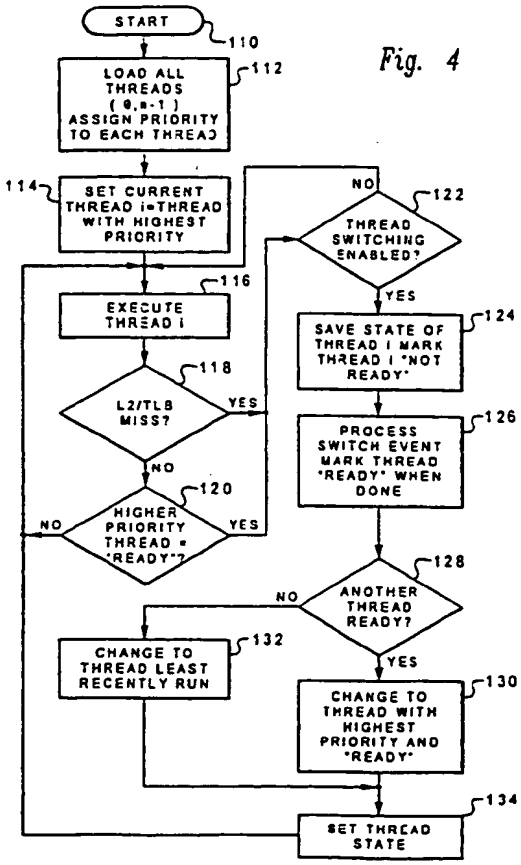

Fig. 4

EP 0 747 816 A2

Description

BACKGROUND OF THE INVENTION

Technical Fi ld

The present invention relates in general to an improved data processing system and in particular to an improved high performance multithread data processing system. Still more particularly the present invention relates to a method and system for reducing the impact of memory latency in a multithread data processing system.

Description of the Related Art

Single tasking operating systems have been available for many years within computer systems. In such systems, a computer processor executes computer programs or program subroutines serially, that is no computer program or program subroutine can begin to execute until the previous computer program or program subroutine has terminated. This type of operating system does not make optimum use of the computer processor in a case where an executing computer program or subroutine must await the occurrence of an external event (such as the availability of data or a resource) because processor time is wasted.

This problem has lead to the advent of operating systems. Each of the program threads performs a specific task. While a computer processor can execute only one program thread at a time, if the thread being executed must wait for the occurrence of an external event, i.e., the thread becomes "non-dispatchable," execution of a non-dispatchable thread is suspended and the computer processor executes another thread of the same or different computer program to optimize utilization of processor assets. Multitasking operating systems have also been extended to multiprocessor environments where threads of the same or different programs can execute in parallel on different computer processors. While such multitasking operating systems optimize the use of one or more processors, they do not permit the application program developer to adequately influence the scheduling of the execution of threads.

Previously developed hardware multithread processors which maintain multiple states of different programs and permit the ability to switch between those states quickly typically switch threads at every memory reference, cache miss or stall. Memory latencies in modern microprocessors are too long and first level on-chip cache sizes are generally quite small. For example, in an object-oriented programming environment program locality is worse than in traditional environments. Such a situation r sults in increas d delays due to incr as d memory access r ndering the data processing system less cost-effective.

Existing multithreading t chniques describe switching threads on a cache miss or a m mory reference. A primary example of this technique may be r viewed in "Sparcle: An Evolutionary Design for Larg -Scale Multiprocessors," IEEE Micro Volume 13, No. 3, pp. 48-60, June 1993. As applied in a so-called "RISC" (reduced instructions set computing) architecture multiple register sets normally utilized to support function calls are modified to maintain multiple threads. Eight overlapping register windows are modified to become four non-overlapping register sets, wherein each register set is a reserve for trap and message handling. This system discloses a thread switch which occurs on each first level cache miss that results in a remote memory request.

While this system represents an advance in the art, modern processor designs often utilize a multiple level cache or high speed memory which is attached to the processor. The processor system utilizes some well-known algorithm to decide what portion of its main memory store will be loaded within each level of cache and thus, each time a memory reference occurs which is not present within the first level of cache the processor must attempt to obtain that memory reference from a second or higher level of cache.

It should thus be apparent that a need exists for an improved data processing system which can reduce delays due to memory latency in a multilevel cache system utilized in conjunction with a multithread data processing system.

SUMMARY OF THE INVENTION

It is therefore one object of the present invention to provide an improved data processing system.

It is another object of the present invention to provide an improved high performance multithread data processor system.

It is yet another object of the present invention to provide an improved method and system for reducing delays due to memory latency in a multithread data processing system.

The foregoing objects are achieved as is now described. A method and system are disclosed for enhanced performance multithread operation in a data processing system which includes a processor, a main memory store and at least two levels of cache memory. At least one instruction within an initial thread is executed. Thereafter, the state of the processor at a selected point within the first thread is stored, execution of th first thread is terminated and a second thread is selected for execution only in response to a level two or higher cache miss, thereby minimizing processor delays due to memory latency. The validity state of each thread is preferably maintained in order to minimize the likelihood of returning to a prior thread for execution before th cache miss has b en corrected. A least recently xecuted thread is preferably selected for x cution in th ev nt of a nonvalidity indication in association with all remaining threads, in anticipation of a chang to the val-

id status of that thread prior to all other thr ads. A thread switch bit may also be utilized to selectively inhibit thread switching where execution of the current thread is deemed necessary.

Th above as well as additional objectives, features, and advantages of the present invention will become apparent in the following detailed written description.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as a preferred mode of use, further objectives and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

Figure 1 is a high level block diagram of a data processing system which may be utilized to implement the method and system of the present invention;

Figure 2 is a high level logic flowchart of a process which may be implemented within the data processing system of **Figure 1** which illustrates basic operation in accordance with the method and system of the present invention;

Figure 3 is a high level logic flowchart of a process which may be implemented within the data processing system of **Figure 1** which illustrates a simple prioritized thread management system in accordance with the method and system of the present invention;

Figure 4 is a high level logic flowchart of a process which may be implemented within the data processing system of **Figure 1** which illustrates a preemptive prioritized thread management system in accordance with the method and system of the present invention;

Figure 5 is a high level logic flowchart of a process which may be implemented within the data processing system of **Figure 1** which illustrates a first thread management system in accordance with the method and system of the present invention; and

Figure 6 is a high level logic flowchart of a process which may be implemented within the data processing system of **Figure 1** which illustrates a second thread management system in accordance with the method and system of the pr s nt inv ntion.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENT

With reference now to the figur s and in particular with refer nce to **Figur 1**, there is depicted a high level block diagram of a data processing system **10** which may be utilized to implement the method and system of the present invention. In a preferred embodiment processor **12** of system **10** is a single integrated circuit superscalar microprocessor, which may be implemented utilizing any well-known superscalar microprocessor system such as the PowerPC Microprocessor manufactured by International Business Machines Corporation of Armonk, New York. As will be discussed below data processing system **10** preferably includes various units, registers, buffers, memories and other sections which are all preferably formed by integrated circuitry. As those skilled in the art will appreciate data processing system **10** preferably operates according to reduced instruction set computing (RISC) techniques.

As illustrated, data processing system **10** preferably includes a main memory store **14**, a data cache **16** and instruction cache **18** which are interconnected utilizing various bus connections. Instructions from instruction cache **18** are preferably output to instruction flow unit **34** which, in accordance with the method and syst m of the present invention, controls the execution of multiple threads by the various subprocessor units within data processing system **10**. Instruction flow unit **34** selectively outputs instructions to various execution circuitry within data processing system **10** including branch unit **26**, fixed point unit **28**, load/store unit **30** and floating point unit **32**.

In addition to the various execution units depicted within **Figure 1** those skilled in the art will appreciate that modern superscalar microprocessor systems often include multiple versions of each such execution unit. Each of these execution units will have as an input source operand information from various registers such as general purpose registers **36** and floating point registers **40**. Additionally, multiple special purpose registers **38** may be utilized in accordance with the method and system of the present invention to store processor state information in response to thread switching.

In a manner well known to those having ordinary skill in the art in response to a load instruction load/store unit **30** will input information from data cache **16** and copy that information to selected buffers for utilization by one of the plurality of execution units. Data cach **16** is preferably a small memory which utilizes high speed memory devices and which stores data which is considered likely to be utilized frequently or in the near future by data processing system. In accordanco with an important feature of the present invention a second lev l cach **20** is also provid d which, in an inclusive syst m, will includ all data stored within data cache **16** and a larger amount of data copi d from main memory stor **14**. Level two cache **20** is preferably a higher speed

memory system than main memory store **14** and, by storing selected data within level two cache **20** in accordance with various well known techniques the memory latency which occurs as a result of a ref rence to main memory store **14** can be minimized.

A level two cache/memory interface **22** is also provided in accordance with the method and system of the present invention. As illustrated, a bus **42** is provided between level two cache/memory interface **22** and instruction flow unit **34** to indicate to instruction flow unit **34** a miss within level two cache **20**. That is, an attempt to access data within the system which is not present within level two cache **20**. Further, a so-called "Translation Lookaside Buffer" (TLB) **24** is provided which contains virtual-to-real address mapping. Although not illustrated within the present invention various additional high level memory mapping buffers may be provided such as a Segment Lookaside Buffer (SLB) which will operate in a manner similar to that described for translation lookaside buffer **24**.

Thus, in accordance with an important feature of the present invention delays due to memory latency within data processing system **10** may be reduced by switching between multiple threads in response to the occurrence of an event which indicates long memory latency may occur. In one embodiment of the system depicted within **Figure 1** a thread switch will occur, if enabled, in response to a level two cache miss on a fetch. That is, an attempt by the processor to access the level two cache to determine whether or not a memory request can be satisfied and an indication that the desired data or instruction is not present within the level two cache. This occurrence is typically processed by causing a memory request to be retrieved from main memory store **14** and the memory latency which occurs during this period triggers, in accordance with the method and system of the present invention, a thread switch. In alternate embodiments of the present invention a thread switch is triggered only in response to the occurrence of those events which will take a longer period of time to complete than is required to refill the instruction pipeline (typically 5 or 6 cycles). Thus, a thread switch may be triggered in response to a Translation Lookaside Buffer (TLB) Miss or Invalidate, a Segment Lookaside Buffer (SLB) Miss or Invalidate, a failed conditional store operation or other operation which require, on average, a period of time which is longer than the time required for a thread switch. By only switching threads in response to such events the necessity for increased complexity and replication of pipeline latches and additional pipeline states is avoided.

A thread is accomplished, as described herein, by providing a thread state register within a dedicated special purpose register **38**. This thread state register preferably includes an indication of the current thr ad number, and indication of whether single-thread or multi-thread operation is enabled and a validity indication bit for each thread. Thus, if four threads are permitted within data processing system **10**, seven bits are required to indicate this information. Additionally, two existing special purpose registers are utilized as save-restore registers to store the address of the instruction which caused the level two cache miss and store the machine state register.

In accordance with the method and system of the present invention level two cache/memory interface **22** preferably permits multiple outstanding memory requests. That is, one outstanding memory request per thread. Thus, when a first thread is suspended in response to the occurrence of a level two cache miss a second thread would be able to access the level two cache for data present therein. If the second thread also results in a level two cache miss another memory request will be issued and thus multiple memory requests must be maintained within level two cache/memory interface **22**. Further, in order to minimize so-called "thrashing" the method and system of the present invention requires that at least a first instruction within each thread must complete. Thus, if all threads within the system are awaiting a level two cache miss and the first thread is resumed it will not find the required data; however, in response to a requirement that at least the first instruction must complete this thread will simply wait until the cache miss has been satisfied.

Thus, those skilled in the art should appreciate that "multithreading," as defined within the present disclosure wherein multiple independent threads are executing may be accomplished in hardware in accordanc with the method and system of the present invention may be utilized to greatly reduce the delay due to memory latency by maintaining the state of multiple threads (preferably two or three in accordance with the current design) and selectively switching between those threads only in response to a second level or higher cache miss.

Referring now to **Figure 2** there is depicted a high level logic flowchart of a process which may be implemented within the data processing system of **Figure 1** which illustrates basic operation in accordance with the method and system of the present invention. As depicted, the process begins at block **60** and thereafter passes to block **62**. Block **62** illustrates the loading of all threads. The process then passes to block **64** which depicts the setting of the current thread i = 0. Block **66** then depicts the execution of thread i until such time as the process passes to block **68**. Block **68** illustrate the occurrence of a level two cache or translation lookaside buffer (TLB) miss. In the event no such miss occurs the process returns, in an iterative fashion, to block **66** to continue to execute thread i.

Referring again to block **68** in the event a level two cache or translation lookaside buffer miss has occurr d, th proc ss pass s to block **70**. Block **70**, in accordanc with an important feature of th present invention, illustrates a determination of whether or not thread switching within the syst m is enabled. Those having ordinary skill

in the art will appreciate that in selected instances execution of a particular thread will be desirable and thus, the method and system of the present invention provides a technique whereby the switching between multiple threads may be disabled. In the event thread switching is not enabled the process passes from block **70** back to block **66** in an iterative fashion to await the satisfaction of the level two cache miss.

Referring again to block **70**, in the event thread switching is enabled the process passes to block **72**. Block **72** illustrates the saving of the state of instruction register and the machine state register for thread i utilizing the special purpose registers (see **Figure 1**) and the process then passes to block **74**. Block **74** illustrates the changing of the current thread to the next thread by incrementing i, accessing the appropriate registers and the process then passes to block **76**. Block **76** illustrates the setting of the thread state for the new current thread and the process then returns to block **66** in an iterative fashion.

With reference now to **Figure 3** there is depicted a high level logic flowchart which illustrates a process which may be implemented within the data processing system of **Figure 1** which depicts a simple prioritized thread management system in accordance with the method and system of the present invention. As illustrated, this process begins at block **80** and thereafter passes to block **82**. Block **82** illustrates the loading of all threads (0, n - 1) and the assignment of an associated priority for each thread. The process then passes to block **84** which depicts the setting of the current thread i equal to the thread having the highest priority. Thereafter, the process passes to block **86**.

Block **86** illustrates the execution of thread i and the process then passes to block **88**. Block **88** illustrates a determination of whether or not a level two cache or translation lookaside buffer miss has occurred and if not, as above, the process returns to block **86** in an iterative fashion to continue to execute thread i.

Still referring to block **88**, in the event a level two cache or translation lookaside buffer miss has occurred the process passes to block **90**. Block **90**, as described above, illustrates a determination of whether or not thread switching is enabled, and if not, the process returns to block **86** in an iterative fashion. However, in the event thread switching is enabled, the process passes to block **92**.

Block **92** depicts the saving of the state of thread i and the marking of that thread as "NOT READY." Thereafter, the process passes to block **94**. Block **94** depicts the concurrent processing of the switch event and the marking of that thread as "READY" when the switch event has been resolved. That is, when the level two miss has been satisfied by obtaining the desired data from main memory store. Continuing, the process passes to block **96**, while processing the switch event as described above, to determine whether or not another thread is ready for execution. If so, the process passes to block **98** which illustrates the changing of the current thread to the thread having the highest priority and a "READY" indication. That thread's thread state is then set, as depicted within block **102** and the process then returns to block **86**, in an iterative fashion as described above.

Referring again to block **96**, in accordance with an important feature of the present invention, in the event another thread within the system does not indicate "READY" the process passes to block **100**. Block **100** illustrates the changing of the current thread to the thread which is least recently run. This occurs as a result of a decision that the thread which was least recently run is the thread most likely to resolve its switch event prior to a subsequent thread and thus, delays due to memory latency will be minimized by selection of this thread as the current thread. The process then passes to block **102** which illustrates the setting of the thread state for this selected thread and the process then returns to block **86** in an iterative fashion.

Referring now to **Figure 4** there is depicted a high level logic flowchart of a process which may be implemented within the data processing system of **Figure 1** which depicts a preemptive prioritized thread management system in accordance with the method and system of the present invention. As illustrated, this process begins at block **110** and thereafter passes to block **112**. Block **112** illustrates the loading of all threads (o, n - 1) and the assignment of an associated priority to each thread. Thereafter, the process passes to block **114**. Block **114** illustrates the setting of the current thread i equal to the thread having the highest priority. The process then passes to block **116** which depicts the execution of thread i.

Next, the process passes to block **118**. Block **118** illustrates a determination of whether or not a level two cache or translation lookaside buffer miss has occurred and if not, the process passes to block **120**. Block **120** illustrates a determination of whether or not a higher priority thread has now been indicated as "READY" and if not, the process returns to block **116** in an iterative fashion, to continue to execute thread i.

Referring again to block **118**, in the event a level two cache or translation lookaside buffer miss has occurred the process passes to block **122**. As described above, block **122** illustrates a determination of whether or not thread switching is enabled and if not, the process returns to block **116** in an iterative fashion. Referring again to block **118**, in the event a level two cache or translation lookaside buffer has not occurred, but, as determined in block **120**, a higher priority thread than the current thread now indicates "READY" the process also passes to block **122**. Block **122** then determines whether or not thread switching is enabled and if not, the process returns to block **116** in an iterative fashion.

Still referring to block **122**, in the event thread switching is enabled, and either a level two cache or translation lookaside buffer miss has occurred, or a

higher priority thread than the current thread now indicates "READY" and thread switching is enabled the process passes to block **124**. Block **134** illustrates the saving of the state of thread i and the marking of that thread as "NOT READY." Next, the process passes to block **126**. Block **126** illustrates the concurrent processing of the switch event, if any, and the marking of the previously current thread as "READY" when that switch event has completed. Of course, those skilled in the art will appreciate that in the event the previously current thread was suspended in response to a higher priority thread indicating a "READY" state no switch event will be processed and the previously current thread will be marked "READY." Next, the process passes to block **128**. Block **128** illustrates a determination of whether or not another thread is ready and, if the process has occurred as a result of a level two cache or translation lookaside buffer miss a determination of the ready state of each thread will be required; however in the event the thread switch occurs as a result of a higher priority thread indicating a "READY" state then the higher priority thread will clearly be available, as determined at block **128**. Thereafter, the process passes to block **130** which illustrates the changing of the current thread to the thread having the highest priority and an indication of "READY."

Alternately, still referring to block **128**, in the event the thread switch has occurred as a result of a level two cache or translation lookaside buffer miss the process passes to block **132**. As described above, block **132** illustrates the changing of the current thread to the thread which was least recently run in accordance with the theory that this thread will be the first thread to achieve a "READY" state. Thereafter, the process again passes to block **134** which illustrates the setting of the thread state for the new current thread and the process then returns to block **116**, in an iterative fashion, as described above.

With reference now to **Figure 5** there is depicted a high level logic flowchart which illustrates a process which may be implemented within the data processing system of **Figure 1** which depicts a first thread management system in accordance with the method and system of the present invention. As depicted, this process begins at block **140** and thereafter passes to block **142**. Block **142** illustrates the loading of an idle loop for each thread (0, n - 1). Next, the current thread is set i = 0, as depicted in block **144**.

The process then passes to block **146** which illustrates the execution of thread i and the process then passes to block **148**. Block **148** illustrates a determination of the occurrence of a switch event while thread switching is enabled. If this occurs the process passes to block **150** which illustrates the switching of threads and the setting of a new current thread. The process then returns to block **146**, in an iterative fashion.

Referring again to block **148**, in the event a determination is made that no switch event has occurred, the process passes to block **152**. Block **152** illustrates a determination of whether or not a task within the current thread has ended and if not, the process returns to block **146** in an iterative fashion to continue execution. However, in the event a task has ended the process passes to block **154**. Block **154** depicts a determination of whether or not another task is ready for execution within the current thread and if so, the process passes to block **156**. Block **156** illustrates the loading of the new task for the current thread and this process then returns, in an iterative fashion, to block **146** to continue execution of the current thread.

Still referring to block **154**, in the event no further tasks are ready within the currently executing thread the process passes to block **158**. Block **158** illustrates the starting of the idle loop for thread i and the process then returns, in an iterative fashion, to block **146** to await the occurrence of one of the enumerated events.

Finally, referring to **Figure 6** there is depicted a high level logic flowchart of a process which illustrates a process which may be implemented within the data processing system of **Figure 1** which depicts a second thread management system in accordance with the method and system of the present invention. As illustrated, this process begins at block **170** and thereafter passes to block **172**. Block **172** illustrates the loading of an idle loop for each thread (0, n - 1). Thereafter, as depicted within block **174**, the current thread i is set = 0. Next, the process passes to block **176**. Block **176** illustrates the marking of the current thread as "VALID" and the marking of all other threads as "NOT VALID." The process then passes to block **178**. Block **178** illustrates the execution of thread i.

Thereafter, as depicted in block **180** in the event a determination is made as to whether or not a switch event has occurred while switching is enabled. If so, the process passes to block **182**. Block **182** indicates a determination of whether or not another thread within the system is "VALID." If not, the process returns to block **178**, in an iterative fashion, to continue execution of thread i. Alternately, in the event another thread is determined as "VALID" The process passes to block **184**. Block **184** illustrates the switching of the current thread to the new thread chosen from among those threads indicating "VALID" state. The process then returns to block **178** to execute the new current thread in the manner described above.

Referring again to block **180** in the event a determination is made that no switch event has occurred or that switching is not enabled, the process passes to block **186**. Block **186** illustrates a determination of whether or not the current task has ended and if so, the process passes to block **188**. Block **188** illustrates a determination of whether or not another task within the current thread is ready for execution and if so, the process passes to block **190**. Block **190** illustrates the loading of the new task for the current thread and the process then returns to block **178**, in an iterative fashion to continue

execution of the current thread.

Referring again to block **188**, in the event a curr nt task has ended, as determined at block **186**, and a subsequent task is not ready the process passes to block **194**. Block **194** illustrates a det rmination of whether or not any other tread within the system indicates "VALID." If not, the process passes to block **196** which illustrates the starting of the idle loop for thread i and the process then returns to block **178**, in an iterative fashion. However, in the event another thread within the system indicates "VALID" the process passes form block **194** to block **200**. Block **200** illustrates the marking of the current thread as "NOT VALID" and the process then returns to block **184** to change the current thread to a new thread chosen from among the valid threads.

Referring again to block **186**, in the event the current task is not ended, the process passes to block **192**. Block **192** illustrates a determination of whether or not a new task has become ready and if not, the process returns to block **178**, in an iterative fashion to continue the execution of thread i in the manner described above. However, in the event a new task has become ready, as determined at block **192**, the process passes to block **198**. Block **198** illustrates a determination of whether or not any "NOT VALID" threads are present among the threads within the system and if not, the process returns to block **178**, in an iterative fashion, to continue to execute thread i. However, in the event a "NOT VALID" thread is present within the system the process passes to block **202**. Block **202** illustrates the selection of one "NOT VALID" thread, the marking of that thread as "VALID" and the loading of the task which is now ready into that thread. The process then returns to block **178**, in an iterative fashion, to continue to execute i. Thereafter, in the event a thread switch event occurs, a "VALID" thread having the new task present therein is ready for execution.

Claims

1. A method for enhanced performance multithread operation in a data processing system which includes a processor, a main memory store and at least two levels of cache memory, said method comprising the steps of:

   executing at least one instruction within a first thread;

   thereafter storing a state of said processor at a selected point within said first thread, terminating execution of said first thread and switching execution to a second thread only in response to an occurr nce of an identifi d vent having a delay associated ther with which xc ds an amount of tim required for a thread switch; and

   executing at least on instruction within said second thread wherein processing delays due to memory access latency are minimized.

2. The method according to Claim 1, further including the steps of:

   storing an indication of non-validity in association with said first thread in response to said occurrence of said identified event, and

   removing said indication of non-validity in association with said first thread following a completion of said identified event.

3. The method according to Claim 1, wherein said data processing system includes a plurality of registers and wherein said step of storing a state of said processor at a selected point within said first thread comprises storing a state of said processor at a selected point within said first thread within a register associated with said first thread.

4. The method according to anyone of Claim 1 to 3, further including the step of determining a validity status for each thread within said data processing system and selecting a second thread for execution in response to said determination,

   selecting a least recently executed thread for execution in response to an indication of non-validity in association with all remaining threads within said data processing system following said occurrence of said identified event, and

   selectively inhibiting execution of a subsequent thread within said data processing system in response to a state of a switch enable bit.

5. The method according to anyone of Claim 1 to 4, further including the step of selecting said second thread for execution following said occurrence of said identified event in response to a priority indication associated with each thread within said data processing system.

6. The method for enhanced performance multithread operation in a data processing system according to anyone of the previous claims

   wherein said switching execution is perform d in response to a level two or higher cache miss; and

   further compris s the step of:
   maintaining an addr ss indication for said l vel two or higher cache miss before xecuting said at least one instruction.

7. A system for nhanced performance multithread operation in a data processing system which include a processor, a main memory store and at least two lev ls of cache memory, said system comprising:

means for executing at least one instruction within a first thread;.

means for thereafter storing a state of said processor at a selected point within said first thread, terminating execution of said first thread and switching execution to a second thread only in response to a level two or higher cache miss;

means for maintaining an address indication for said level two or higher cache miss; and

means for executing at least one instruction within said second thread wherein processing delays due to memory access latency are minimized.

8. The system according to Claim 7, further including:

means for storing an indication of non-validity in association with said first thread in response to said level two or higher cache miss,

means for removing said indication of non-validity in association with said first thread following a retrieval from main memory of data or instruction at said maintained address indication for said level two or higher cache miss.

9. The system according to Claim 7, wherein said data processing system includes a plurality of registers and wherein said means for storing a state of said processor at a selected point within said first thread comprises storing a state of said processor at a selected point within said first thread within a register associated with said first thread.

10. The system according to Claim 7, 8 or 9 further including:

means for determining a validity status for each thread within said data processing system and selecting a second thread for execution in response to said determination,

means for selecting a least recently executed thread for execution in response to an indication of non-validity in association with all remaining threads within said data processing system following said lev l two or higher cache miss,

means for selectively inhibiting execution of a subsequent thread within said data processing system in response to a state of a switch enable bit, and

means for selecting said second thread for execution following said level two or higher cache miss in response to a priority indication associated with each thread within said data processing system.

11. A computer program product comprising a system for enhanced multithread operation in a data processing system according to any one of claims 7 to 10.

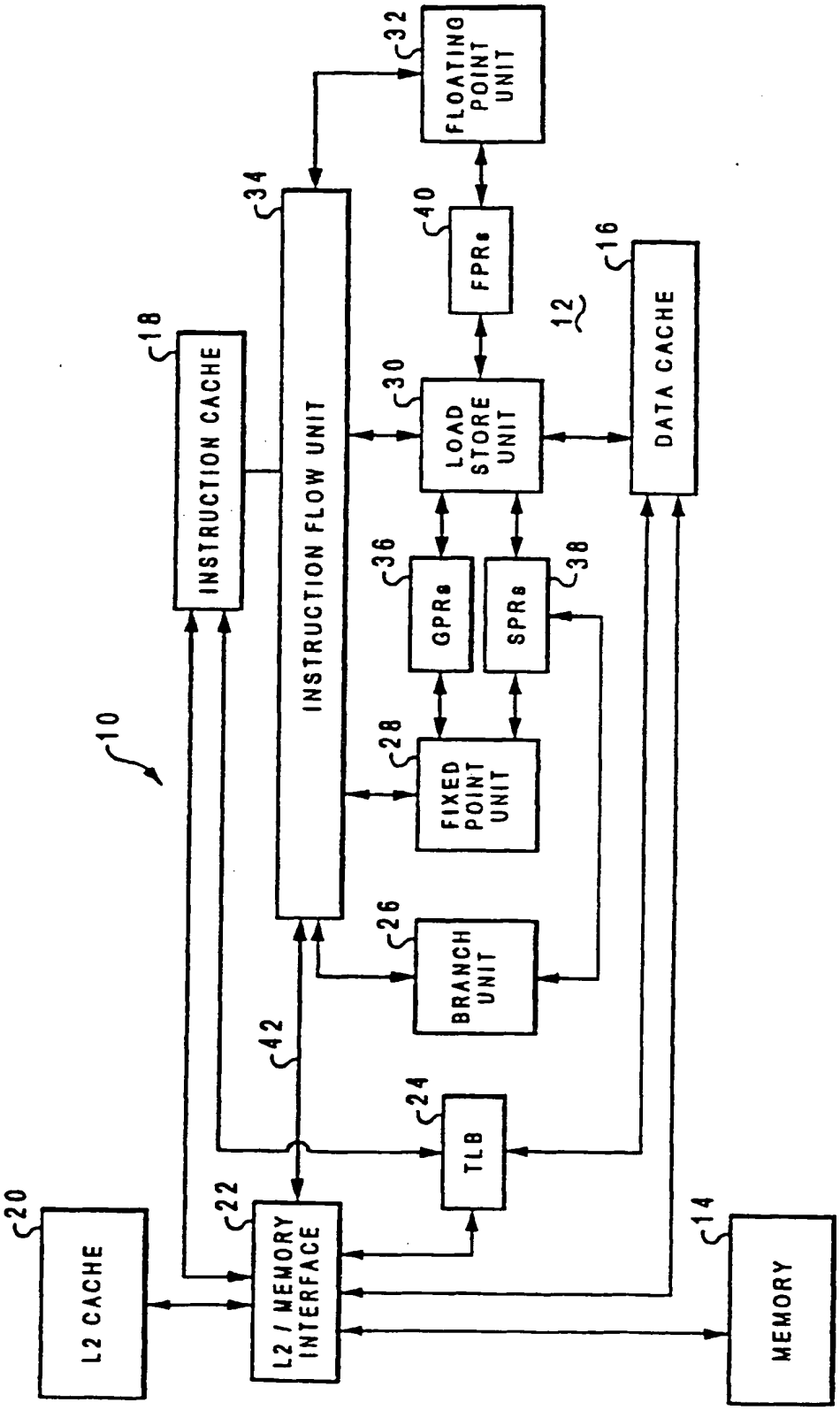

Fig. 1

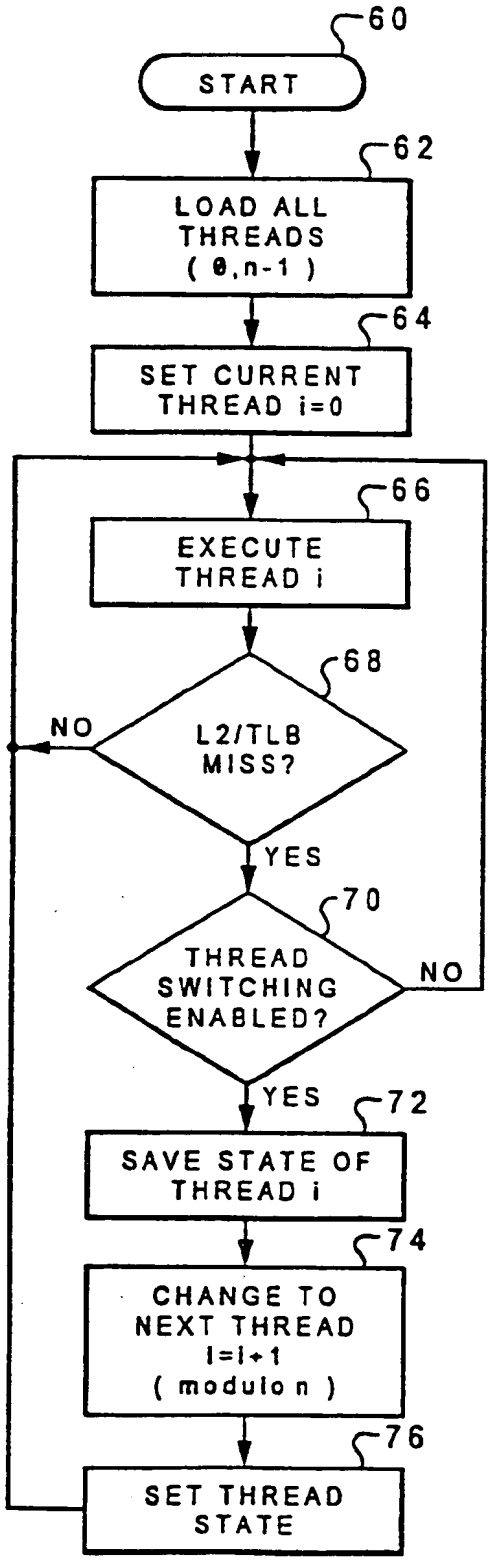

Fig. 2

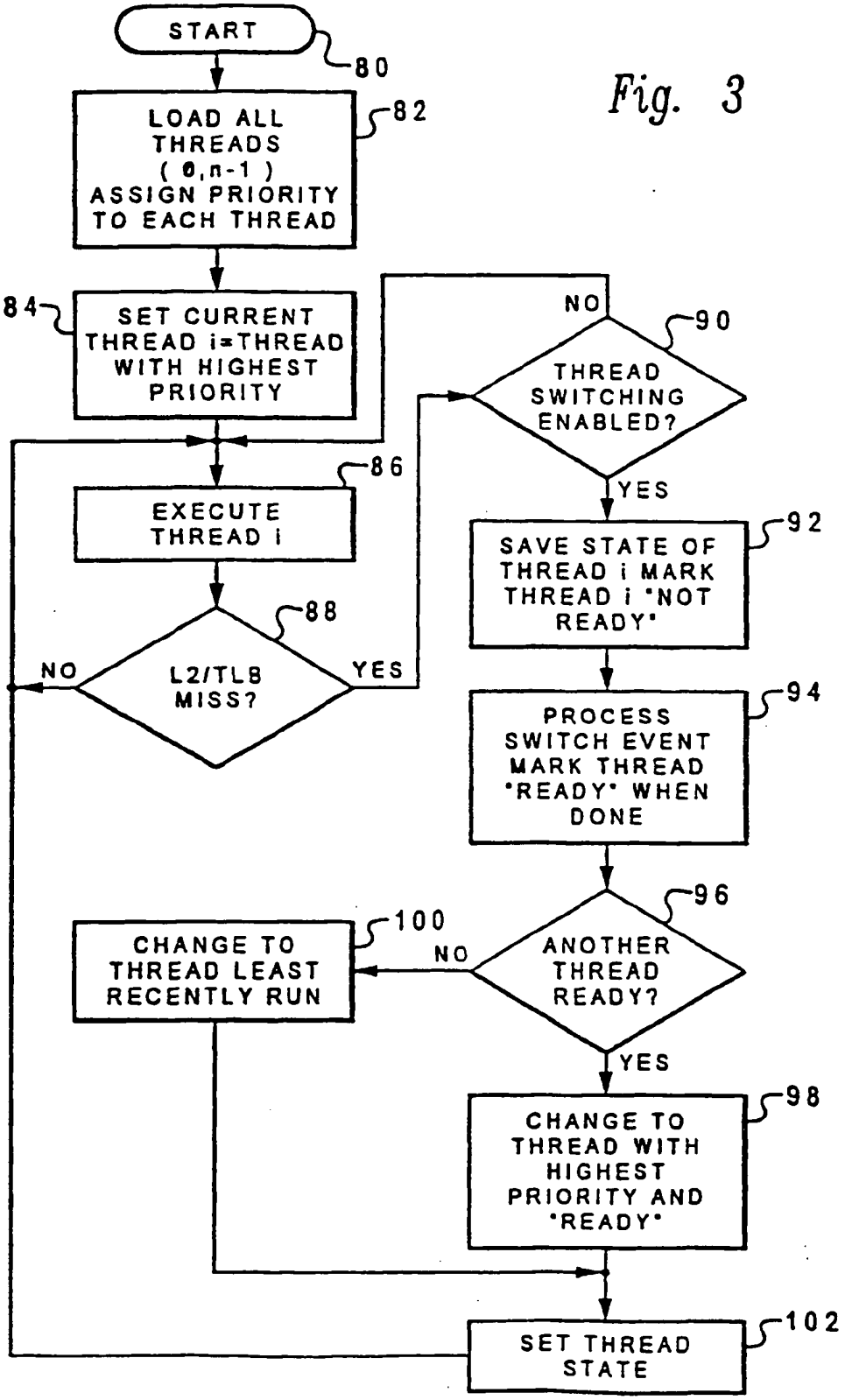

Fig. 3

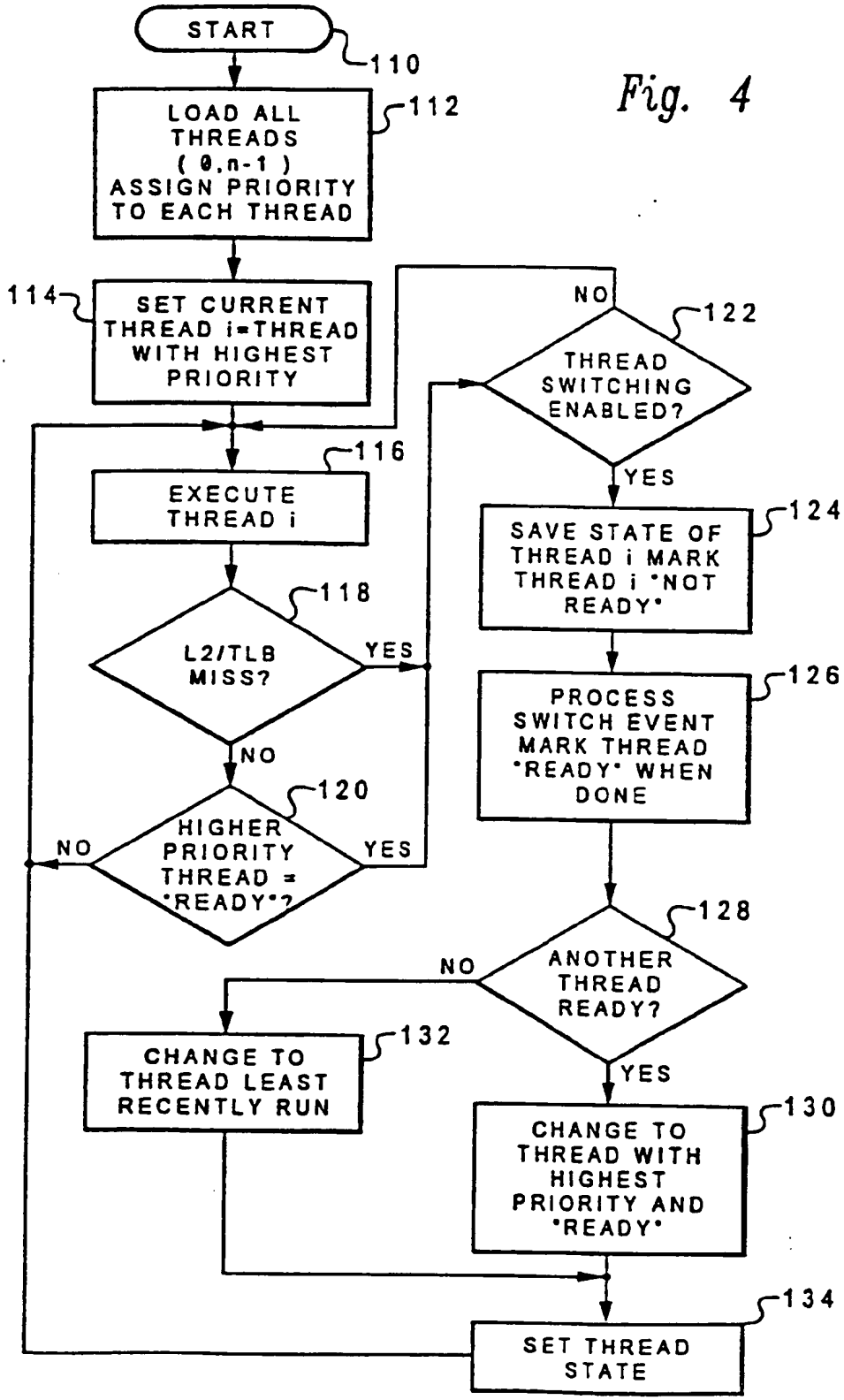

Fig. 4

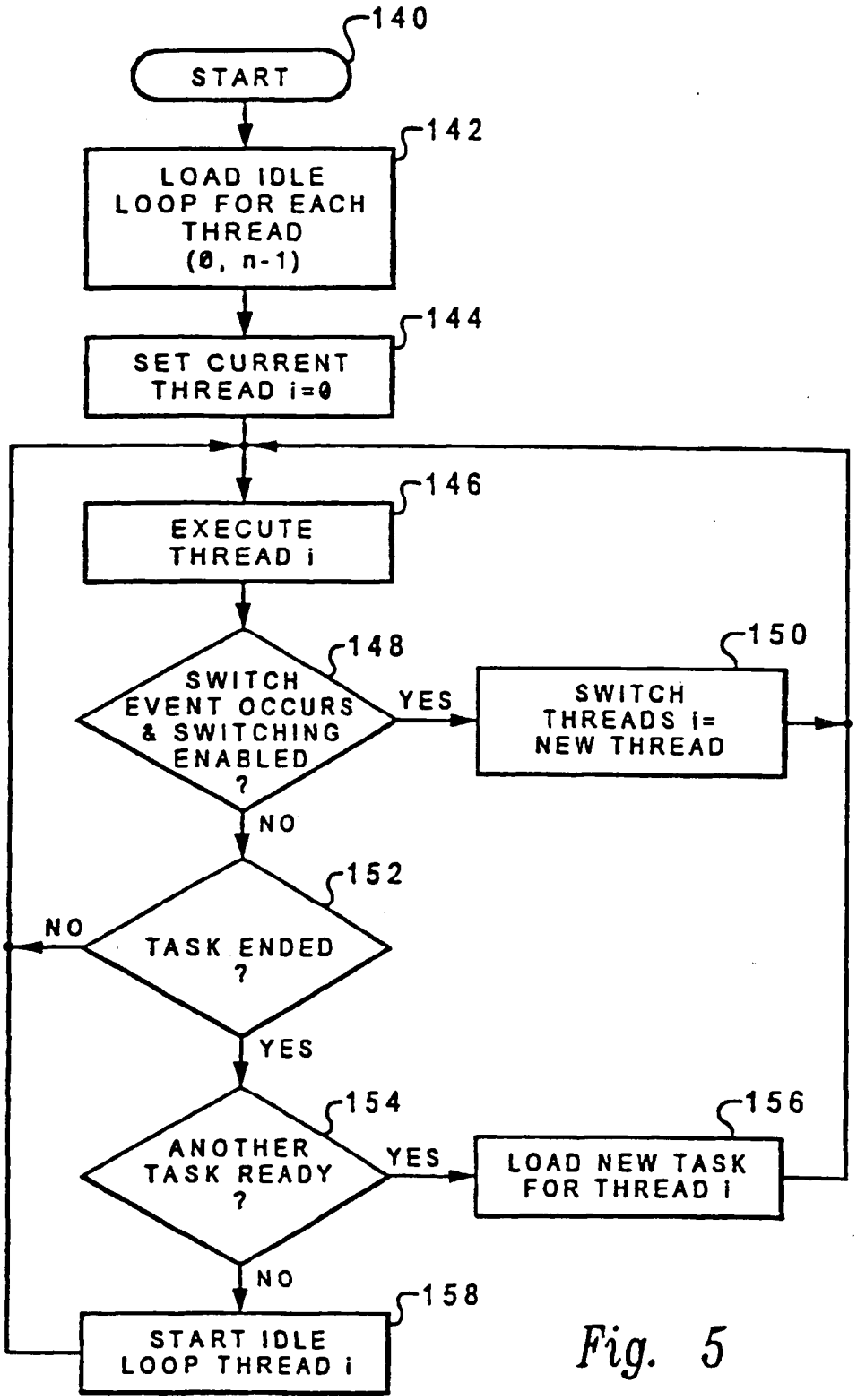

Fig. 5

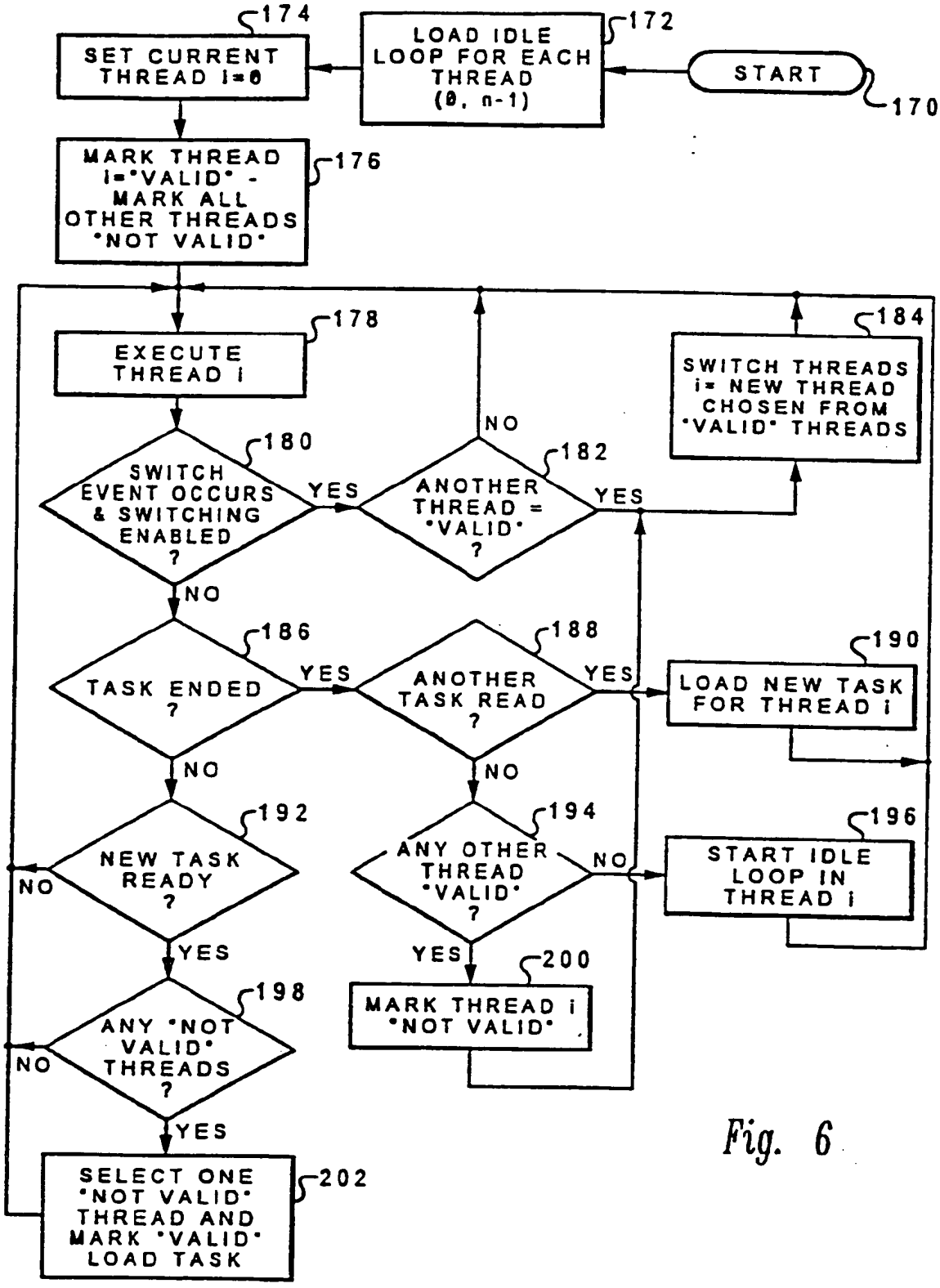

Fig. 6